

1. State Machines and Finite State Automata

Contents

- Alphabets and strings
- Languages and machines
- Deterministic FSA and transition diagrams
- Non-deterministic FSA
- Comparing DFSA and NFSA

Alphabets

Alphabet Σ : Finite set of symbols/characters used for encoding

Examples:

- $\Sigma = \{ 0, 1 \}$
- $\Sigma = \{ a, b \}$
- $\Sigma = \{ A, C, G, T \}$
- $\Sigma =$ All characters in the Roman alphabet

Strings

A string w over an alphabet Σ is a finite sequence of symbols in Σ

Example strings over the alphabet $\Sigma = \{ a, b \}$

- ε : empty string, i.e., string of length 0
- a
- b
- abb
- baaabba

Σ^* = Set of all strings over Σ

If Σ has m symbols, how many strings of length k ?

Languages and Machines

- A **language** is defined as a set of strings.
- A language L over an alphabet Σ is a set of strings over Σ .
 L is a subset of Σ^*
- The **machines** to be considered are those that **compute** by:
 - **accepting** **all** the strings in some language, *and*
 - **rejecting** **all** others.
- Equivalence between language and machine: accept words if and only if part of the language

Languages and Machines (cont.)

Machine M for a language L :

Given an input string w , machine M scans w from left to right,
processing one symbol in each step

Reading the string w , M should either accept or reject w , depending
on whether or not w is in L

Languages and Problems

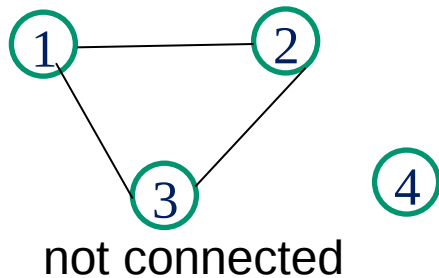
- (Again..) A language L over an alphabet Σ is a set of strings over Σ . L is a subset of Σ^*
- A problem is encoded as a language L :
Given an input string w , decide whether or not w is in L
- This way we can encode all *decision problems*, i.e., problems where the answer is either yes or no

Languages and Problems: Example 0

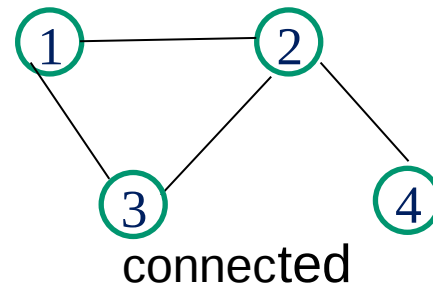
$L = \{ \langle G \rangle \mid G \text{ is a connected graph} \}$

$\langle G \rangle$ is an encoding of graph G as a string over some appropriate alphabet

$\langle G_A \rangle = (1,2,3,4)((1,2),(1,3),(2,3))$



$\langle G_B \rangle = (1,2,3,4)((1,2),(1,3),(2,3),(2,4))$



Decision Problem: Given $\langle G \rangle$, is G in L ?

Example Language 1:

$$\Sigma = \{ a, b \}$$

$$\begin{aligned} L &= \{ w \mid w \text{ ends with the symbol } a \} \\ &= \{ a, aa, ba, aba, baa, aaa, bba, \dots \} \end{aligned}$$

A machine for L needs to check, given an input string w , whether or not the last symbol of the input w is a

Example Language 2:

- The set of all words of the form: $a^m b^n c^{m+n}$, where
 - m, n range over the numbers $0, 1, 2, \dots$
 - x^n denotes the string containing n successive copies of the symbol x .
- That is, $\{a^m b^n c^{m+n} \mid m, n \geq 0\}$ and **alphabet** $\Sigma = \{a, b, c\}$.
- Here are some **valid** words in this language:
 - aaabbccccc aabbccccc ac
- Some words with the same alphabet but **not** in this language:
 - aabbcc abccc bbaccc
- This language might be said to ‘represent’ **addition**.

Addition and subtraction

- Similarly, $\{a^m b^n c^{m-n} \mid m \geq n \geq 0\}$ containing:
– aaaabccc aaaaabbbbc
is a representation of **subtraction**.
- Many other functions may be represented in the form of a language.
- A **machine** that accepts only words in Language 2, and rejects all others, could be described as an addition **acceptor**.

Example Language 3

$$\Sigma = \{ 0, 1 \}$$

$$\begin{aligned} L &= \{ w \mid w \text{ has an odd number of 1s} \} \\ &= \{ 1, 01, 10, 001, 111, 010, 1011, \dots \} \end{aligned}$$

- We will see a machine that accepts L shortly.

Classes of Language/Machine: Regular Languages

- Language 3 belongs to the class of **regular languages**.
- Language 2 does not.
- For every **regular language**, an acceptor machine can be constructed that belongs to the **class of machines** called **finite state automata**.
- (*Note: one automaton, many automata.*)

Classes of Language/Machine: Context-free Languages

- No finite state automaton can be designed to accept the languages of addition and subtraction.
- They belong to the class of **context-free languages** (a class that includes most programming languages).
- To recognise a context-free language requires the power of the class of machines called **push-down automata**.
- Note that push-down automata can also recognise the regular languages.

Classes of Language/Machine: Context-sensitive Languages

- The push-down automata are also limited in their **computational power**, that is,
 - in the class of languages they can recognise,
- or equivalently,
 - in the class of functions they can compute.
- They cannot recognise the **context-sensitive languages**, which require the power of **Turing Machines**.
E.g. $\{a^n b^m a^n b^m \mid m, n \geq 0\}$
- This class of machine is the most powerful.
In fact, no function that cannot be computed by a Turing Machine can be computed at all!

Finite state automata (informal)

- A finite state automaton (FSA) or just finite automaton consists of
 - a finite set of **states** and
 - a finite set of **transitions** from state to state, each of which is associated with a symbol from an **alphabet**.
- Transitions tell us, for each state, based on symbol being processed, what should the next state be
- One state is the **initial** state, in which the automaton starts.
- Some (or none) states are designated as **accept/final** states.
- After reading the input, if the automaton ends up in an accept state, then it accepts the input; otherwise, if stuck at some point, or if ending in a state that is not an accept state, it rejects the input.

Deterministic finite state automata (formal)

- A **Deterministic Finite State Automaton** is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, where:
 - Q is a finite set of **states**
 - Σ is a finite input **alphabet** (set of symbols)
 - δ is the **transition function**, $Q \times \Sigma \rightarrow Q$, that determines, for **each state and for each symbol** of the alphabet, the next state.
 - q_0 is the **initial state** (or start state), $q_0 \in Q$
 - F is the set of **accept/final states** ($F \subseteq Q$).
- One (and only one) transition exists for each pair of state and symbol
- The initial state can be also an accept state

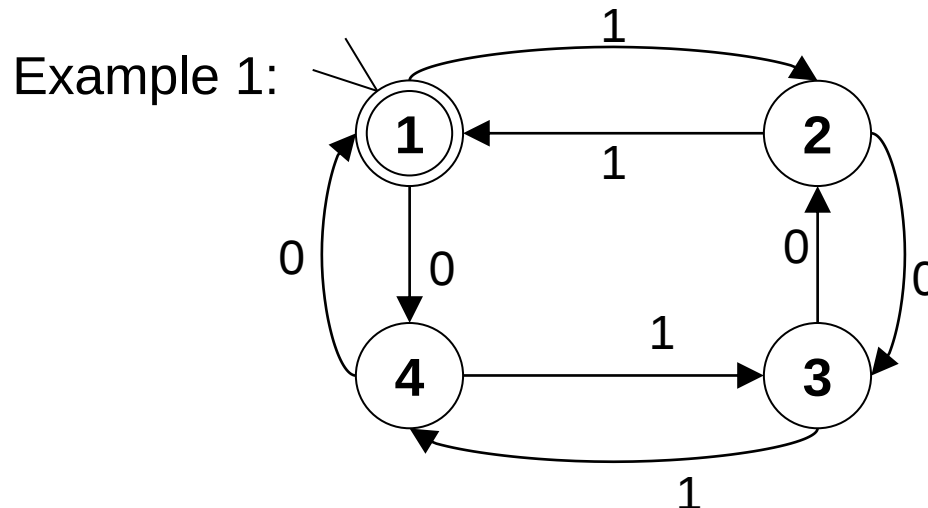
Deterministic finite state automata

- Computation proceeds starting in the initial state, using up one input symbol at a time and following the appropriate transition, until the input is used up. Note: **All input must be read !**
- If the automaton is in one of the accept states (also called favourable states) when the string of input symbols comes to an end, then the input is **accepted**, or recognised (as a member of the language to be recognised), otherwise it is **rejected**.
- For an FSA $M = (Q, \Sigma, \delta, q_0, F)$, we define the language $L(M)$ of M as

$L(M)$ = the set of all strings over Σ accepted by M .

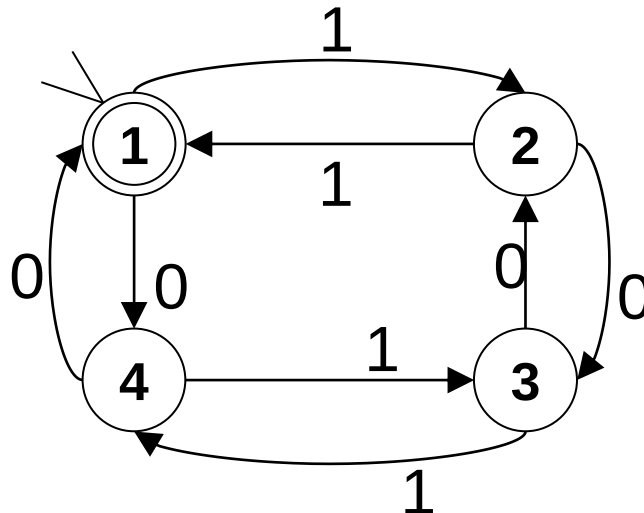
Transition diagrams

- Any FSA may be represented by its **transition (or state) diagram**.
- This is a directed graph in which
 - each state is denoted by a circle
 - each transition is denoted by an arrow between two states, labelled by the associated symbol,
 - the start state is indicated by an incoming arrowhead
 - the accepting state(s) are denoted by a double circle.



Example 1 (cont.)

- This FSA has alphabet, $\{0,1\}$ and four states, numbered 1, 2, 3 and 4.
- State 1 is both the initial state and the only accept state.



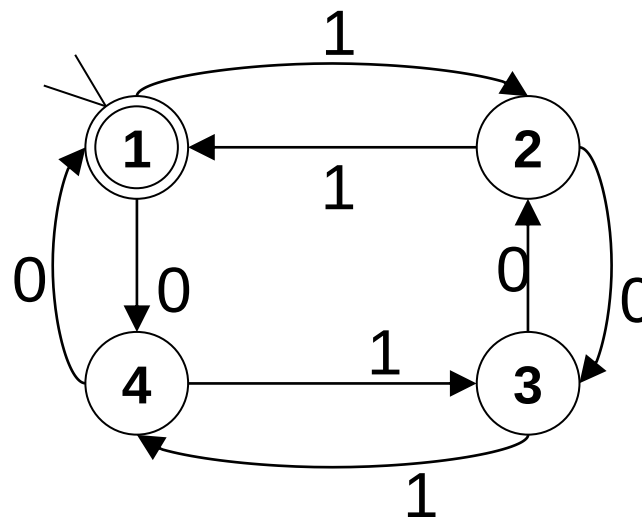
Do not confuse the names of the states with the symbols in the alphabet.

Transition function: $\delta(1,0) = 4$, $\delta(1,1) = 2$, $\delta(2,0) = 3$, $\delta(2,1) = 1$, $\delta(3,0) = 2$, $\delta(3,1) = 4$, $\delta(4,0) = 1$, $\delta(4,1) = 3$

(Note: In mathematical proofs we may sometimes use symbols such as q_1 , $q_2, \dots$ for states)

Example 1 (cont.)

- Its behaviour when presented with a string of symbols, say, '1010', may be traced:
 - start in state 1;
 - read the first symbol in the string ('1') and follow the edge labelled with that symbol, going to state 2;
 - read the next symbol in the string ('0') which leads to state 3;
 - the next symbol ('1') takes it to state 4;
 - the last symbol of the string ('0') takes it to state 1, which is an accept state.
- The word '1010' is therefore in the language accepted by this FSA.



Example 2

- Consider the language L that comprises of:
 - the string containing only the single symbol '0' and every string of symbols from the alphabet $\{0,1\}$ that begins with '1'.
- The FSA that recognises L has the following values:

$$Q = \{1,2,3,4\}$$

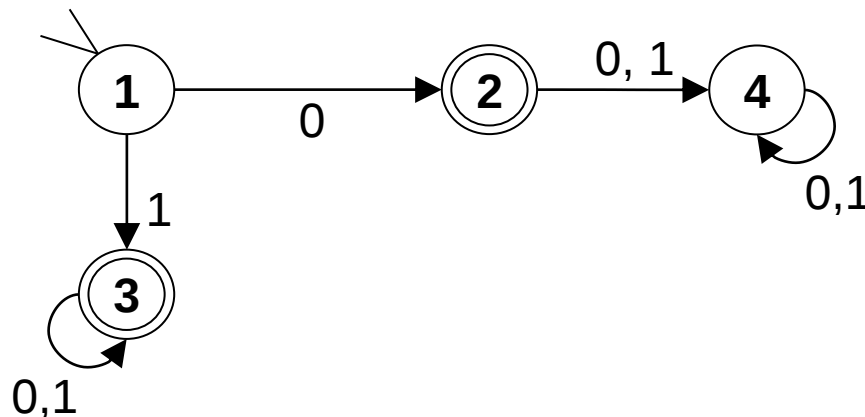
$$\Sigma = \{0,1\}$$

$$q_0 = 1 \text{ (initial state)}$$

$$F = \{2,3\}$$

δ is defined by the table:

	0	1
1	2	3
2	4	4
3	3	3
4	4	4



Example 3

$$\Sigma = \{0, 1\}$$

$$L = \{w \mid w \text{ is a string with an odd number of } 1\text{s}\}$$

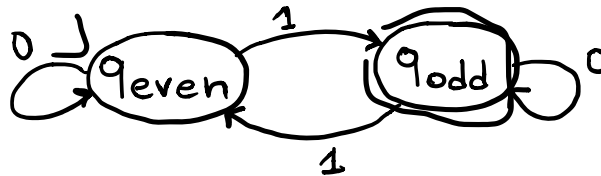
- Construct an FSA that recognizes L

Example 3 (cont.)

$$\Sigma = \{0, 1\}$$

$$L = \{w \mid w \text{ is a string with an odd number of 1s}\}$$

- Construct an FSA that recognizes L
- Pretend you are the automaton :
 - What is the necessary information you need to remember about the input string as you read it?
 - It is whether the number of 1s read so far is even or odd: one state for each possibility

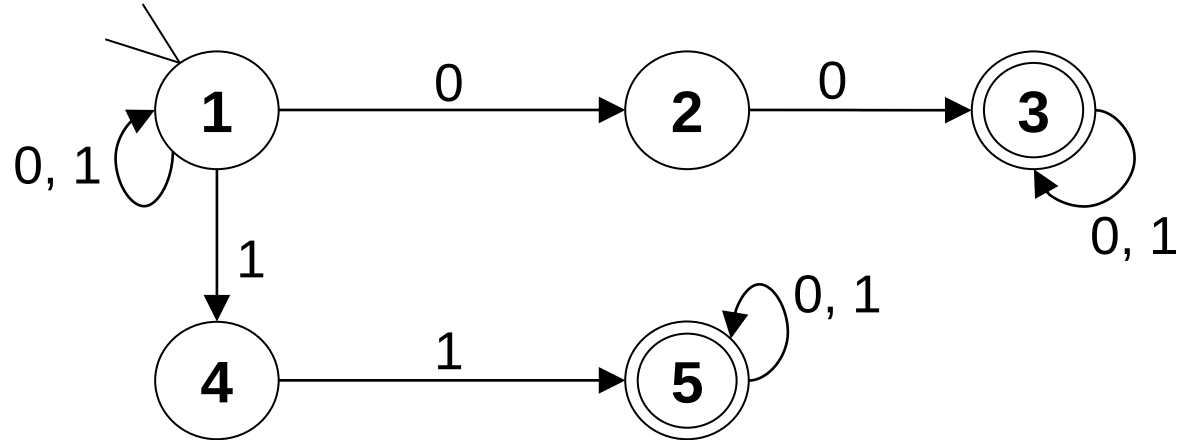


- keep track of this information as you read new symbols : switch state if you read 1

Non-Deterministic Automata

- For the deterministic FSAs (DFSA) above, state transitions are defined by a **function**, that is, at each state there is **exactly one** outgoing edge for each symbol.
- If **more than one outgoing** edge from any state is labelled with the same symbol, the FSA is **non-deterministic** (NFSA).
- Also, in an NFSA, a state **may have no outgoing** edges for some symbol.
- And, in an NFSA, an outgoing edge may be labelled with ε (no string symbol required) – the machine reads no input in this case when transitioning.
- A string is accepted by a NFSA if there exists **at least one** sequence of transitions, corresponding to the input string, leading from the start state to an accept state.
- You can think of the NFSA as, for an input, running in parallel all possible sequences of transitions deterministically

Example



- Note that there are two outgoing arcs from state 1 labelled with the symbol '0'.
- Words '01001' and '11001' are both accepted by this NFSA as there are sequences of transitions (arcs) end in an accept state labelled by 0,1,0,0,1 and 1,1,0,0,1 respectively.

Non-deterministic finite state automata (formal)

- A **Non-deterministic Finite State Automaton** (NFSA) is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, where:
 - Q is a finite set of **states**
 - Σ is a finite input **alphabet** (set of symbols)
 - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the **transition relation**, that determines, for each state and for each symbol of the alphabet (or ε), the possible next states.

$$\Sigma_\varepsilon \equiv \Sigma \cup \{\varepsilon\}$$

$P(Q) \equiv$ collection of all subsets of Q , the *power set* of Q

The transition function is $Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$

note: $P(Q)$ includes the empty set $\emptyset$ (no transitions)

- q_0 is the **initial state** (or start state), $q_0 \in Q$
- F is the set of **accept states** ($F \subseteq Q$).

Non-deterministic finite state automata

- If amongst all the possible states that can be reached from the start state by reading the complete input string w there is an accept state, then the NFSA **accepts** w , otherwise it **rejects** it.
- Note one big difference, *NFSA do not require that every state has a transition for every letter of the alphabet.*
- If a state has no transition for the next symbol in a string, then the string cannot be accepted by extending computational path on that sequence of states, and reading in that sequence is aborted.

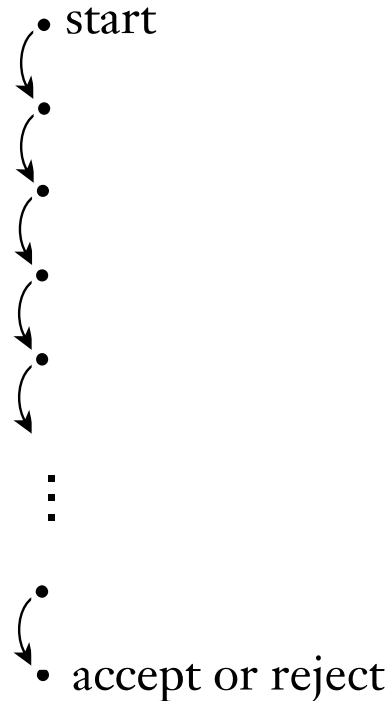
Computational paths

- “Parallel computation”
(threads)

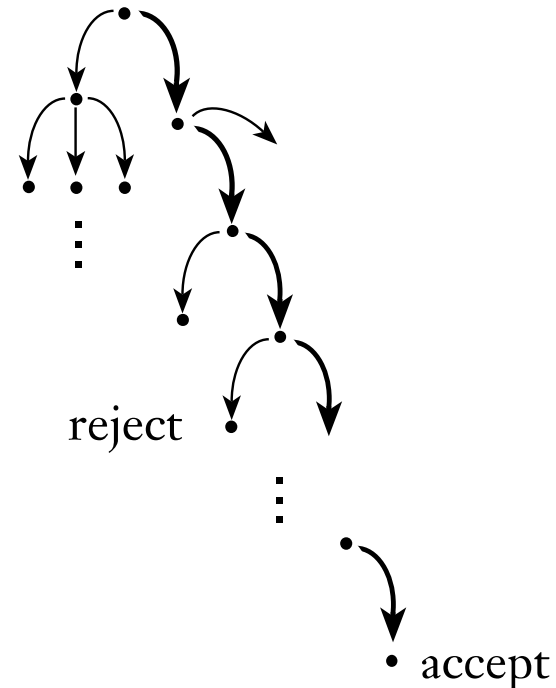
or

- Tree of “possibilities”

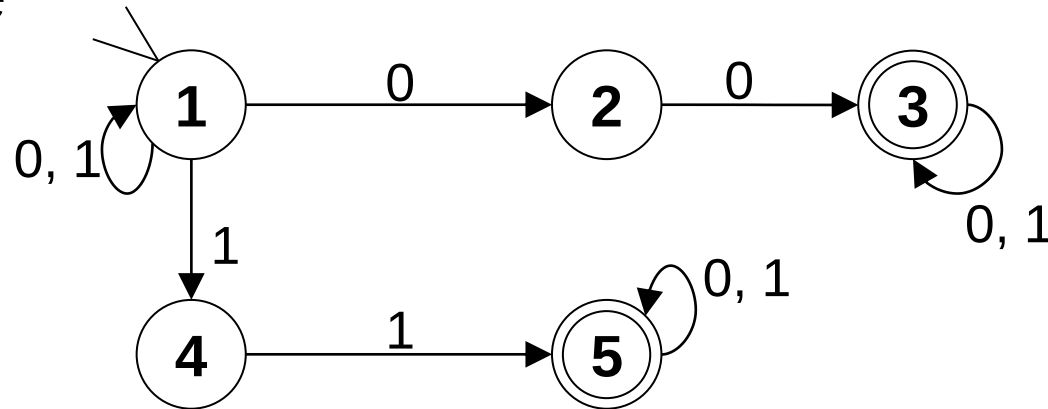
Deterministic
computation



Nondeterministic
computation



Example



- The state transition relation for this NFSA can be viewed as a map from each state-symbol pair to a **subset of states** (that may be empty) among which the NFSA may freely choose.

	0	1
1	{1,2}	{1,4}
2	{3}	$\emptyset$
3	{3}	{3}
4	$\emptyset$	{5}
5	{5}	{5}

•The language recognised by this NFSA comprises all strings over the alphabet $\{0,1\}$ that contain either two successive 0s or two successive 1s.

Null transitions

- A NFSA may also have edges labelled with **the empty symbol**, denoted by ' ϵ '.
- Note that this is **not a symbol in the alphabet** of the NFSA. It labels **null transitions**: edges that the NFSA may follow **without reading the symbol on its input tape**.
- If an NFSA is in a state from which a null transition leaves, then it may **choose**, non-deterministically, to follow that edge to the next state.

$DFSA \subseteq NFSA$

- Languages recognized by some DFSA are also recognized by some NFSA
- Every deterministic FSA (DFSA) is representable as an NFSA.
- **Proof:** Simply, the transition relation of the NFSA is the transition function of the DFSA (if you can have 0,1,or more transitions for each pair state-symbol, this includes the option of having exactly 1)

DFSA $\subseteq$ NFSA

- Every deterministic FSA (DFSA) is representable as an NFSA.
- **Proof:** Simply, the transition relation of the NFSA is the transition function of the DFSA.
- Therefore, **every language recognisable by a DFSA is also recognisable by a NFSA.**
- This means that the class of DFSAs is **no more powerful** than the class of NFSAs.

NFSA $\subseteq$ DFSA ?

- Is the **converse** true? If there was a language recognisable by a NFSA but not by any DFSA, then the class of NFSAs would be more powerful than the class of DFSAs.

NFSA $\subseteq$ DFSA ?

- Is the **converse** true? If there was a language recognisable by a NFSA but not by any DFSA, then the class of NFSAs would be more powerful than the class of DFSAs.
- Yes: To **prove** that NFSA $\subseteq$ DFSA, we give a **constructive algorithm** that converts any NFSA, M , to its equivalent DFSA, M' .

NFSA $\subseteq$ DFSA

- The NFSA, M , occupies, at any moment, not a single state but the **set** of all states that can be reached from the initial state by means of the input consumed so far.
- The **state set**, $Q(M') \subseteq \wp(Q(M))$ (i.e., the powerset of $Q(M)$)
- The **initial state**, $q_0(M') = \{q \in Q(M) \mid q \text{ is reachable from } q_0(M) \text{ by null transitions only}\} \cup \{q_0(M)\}$
- The **accept states**, $F(M') = \{a \in Q(M') \mid a \cap F(M) \neq \emptyset\}$

NFSA $\subseteq$ DFSA

- A move of M' on reading an input symbol imitates a move of M on reading that symbol, *possibly **followed by any number of null transitions of M .***
- Define $E(q)$ to be, for each state q in Q , the set of states that can be reached from q without reading any input (null transitions). That is, states that can be reached through one **or more** null transitions (including q itself).

NFSA $\subseteq$ DFSA

- Then the **transition function**, $\delta(M')$, is derived from $\delta(M)$ by calculating, **for every subset of $Q(M)$ (that is a state of M')**, and **for each symbol in the alphabet**, the set of states that can be reached from any member of the subset by reading that symbol and applying E to the destination states. $Q(M') \times \Sigma \rightarrow Q(M')$

NFSA $\subseteq$ DFSA

- Then the **transition function**, $\delta(M')$, is derived from $\delta(M)$ by calculating, **for every subset of $Q(M)$ (that is a state of M')**, and **for each symbol in the alphabet**, the set of states that can be reached from any member of the subset by reading that symbol and applying E to the destination states. $Q(M') \times \Sigma \rightarrow Q(M')$

$$Q' = \mathcal{P}(Q)$$

$$q'_0 = E[q_0]$$

$$F' = \{q' \in Q' \mid q' \text{ contains a state } q \text{ that is an accept state}\}$$

$$\delta(q', a) = \{q_i \in Q \mid q_i \in E[\delta(q_j, a)] \text{ for some } q_j \in q'\}$$

NFSA $\subseteq$ DFSA

- Proof by construction: A DFSA sequence of states keeps track of all parallel sequences of the NFSA

$$Q' = \mathcal{P}(Q)$$

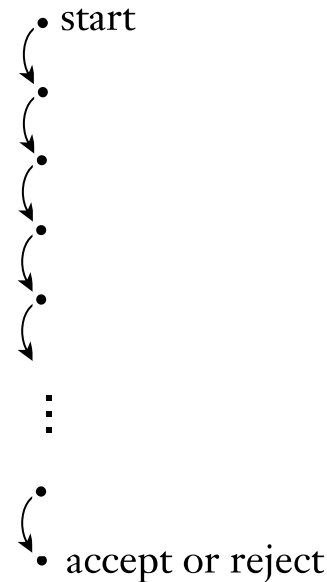
$$q'_0 = E[q_0]$$

$$F' = \{q' \in Q' \mid q' \text{ contains a state } q \text{ that is an accept state}\}$$

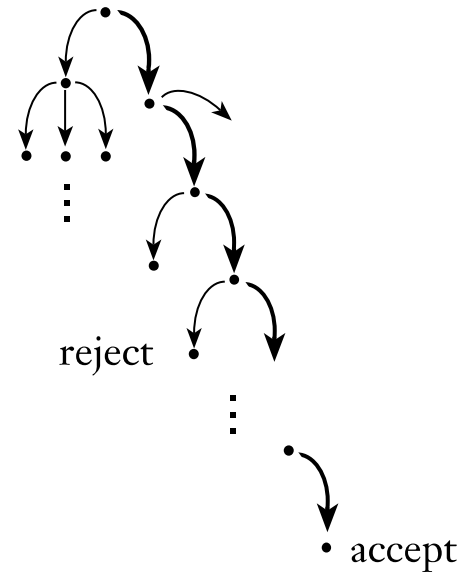
$$\delta(q', a) = \{q_i \in Q \mid q_i \in E[\delta(q_j, a)] \text{ for some } q_j \in q'\}$$

- Since this algorithm may be performed on any NFSA, it can be conclude that
- **NFSA = DFSA**

Deterministic computation



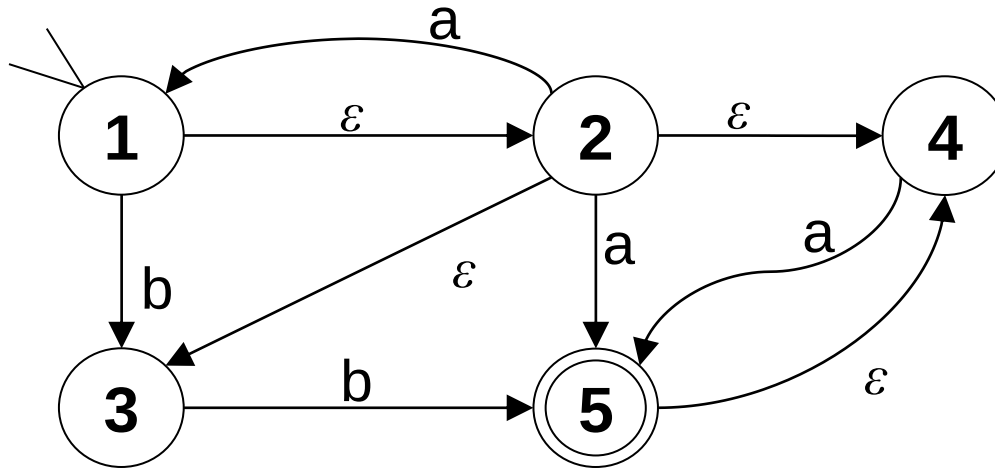
Nondeterministic computation



Summary of NFSA to DFSA

- Define a start state consisting of all the states reachable from NFSA start state using null transitions
(Note: include the NFSA start state!)
- **New states are sets**
- Build a transition: take each element of the set, find states reachable by using the character, **then null**; take the union of these
- New row of the transition table for each new set
- Don't forget the **empty set** (the empty set is just a name of a state of the DFSA)
- Accept states are those sets containing in their name an accept state of the NFSA

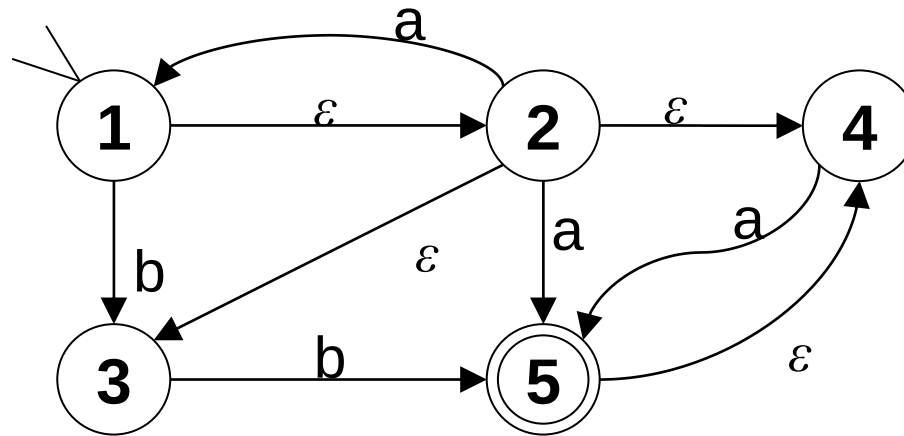
Example



- This has the following state transition table:

	a	b	ε
1	$\emptyset$	$\{3\}$	$\{2\}$
2	$\{1,5\}$	$\emptyset$	$\{3,4\}$
3	$\emptyset$	$\{5\}$	$\emptyset$
4	$\{5\}$	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$	$\{4\}$

Example

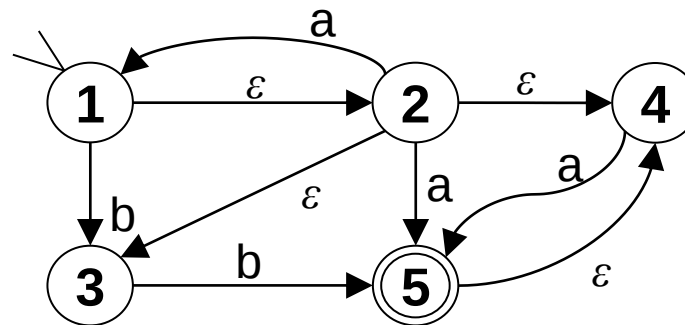


- To transform the above to a DFSA:
- First, enumerate the function E:
 - $E(1) = \{1,2,3,4\}$, $E(2) = \{2,3,4\}$, $E(3) = \{3\}$, $E(4) = \{4\}$, $E(5) = \{4,5\}$
- Then identify the start state of the DFSA by applying E to the start state of the NFSA.
- In this case, $E(1) = \{1,2,3,4\}$, so the new start state is $\{1,2,3,4\}$.

Example

- Then identify, successively, for each new state, the set of states reachable by the NFSA for each symbol in the alphabet, and check whether each of these is a start and/or accept state of the new machine, thus:

NFSA:



DFSA:

	new states	a	b
(start)	{1,2,3,4}	{1,2,3,4,5}	{3,4,5}
(accept)	{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
(accept)	{3,4,5}	{4,5}	{4,5}
(accept)	{4,5}	{4,5}	∅
	∅	∅	∅

Example

- Then (optionally) rename the new states:

			a	b
6	{1,2,3,4}	(start)	7	8
7	{1,2,3,4,5}	(accept)	7	8
8	{3,4,5}	(accept)	9	9
9	{4,5}	(accept)	9	10
10	∅		10	10

- This is, by inspection, a function, and has no null transitions, hence new machine is a DFSA.
- (Check that it recognises the same language as the NFSA.)

Summary

- FSA is a model of computation.
- DFSA and NFSA have equal computational power.
- This can be demonstrated by giving an algorithm to translate any NFSA to a DFSA accepting the same language (and vice versa, trivially).

Important: Work through the exercises in the labs (available on Moodle)!

Reading

- From Sipser:
- Check the mathematical preliminaries, pages 1-25.
- Read sections 1.1 and 1.2, pages 29-63.
- Try exercises 1.1-1.10. Try 1.11 if you want a challenge